Real-Time Systems

Lecture 4

Scheduling basics

Task scheduling - basic taxonomy
Basic scheduling techniques
Static cyclic scheduling
Last lecture (3)

Real-time kernels

- The task states
  - States and transition diagram
- The generic architecture of a RT kernel
- The basic components of a RT kernel, its structure and functionalities
- Some examples: RTKPIC18, SHaRK and XENOMAI
Temporal complexity

- Measurement of the **growth** of the **execution time** of an algorithm as a function of the **problem size** (e.g. the number of elements of a vector, the number of tasks of a real-time system)
- Expressed via the **O( ) operator (big O notation)**
- **O( ) arithmetic**, n=problem dimension, k=constant
  - \( O(k) = O(1) \)
  - \( O(kn) = O(n) \)
  - \( O(k_1n^m+k_2n^{m-1}+...+k_{m+1}) = O(n^m) \)

```plaintext
for (k=0;k<N;k++)
a[k]=0;
Compl. = O(N)
```

```plaintext
for (k=0;k<N-1;k++)
  for (m=k;m<N;m++)
    if a[k]<a[m]
      swap(a[k],a[m]);
Compl. = O(N^2)
```

Computation of the permutations of a set

A={a_i, i=1..N}

Compl. = O(N^N)

---

- **Expressed via the O( ) operator (big O notation)**
- **O( ) arithmetic**, n=problem dimension, k=constant
  - \( O(k) = O(1) \)
  - \( O(kn) = O(n) \)
  - \( O(k_1n^m+k_2n^{m-1}+...+k_{m+1}) = O(n^m) \)
Temporal complexity

What is the complexity of scheduling tasks?

• Build all possible schedules with two tasks.
  - E.g.
    - \{1,2\}
    - \{2,1\}

• Build all possible schedules with 3 tasks.

• Build all possible schedules with 4 tasks.

Comments ...
Temporal complexity

P and NP classes in decision problems

- **P** – problem that can be solved in polynomial time, \( O(p(N)) \)

- **NP** – problem that cannot be solved in polynomial time but for which a solution can be tested in polynomial time
  
  - **NP-complete**
    
    - No “quick” solutions are known.
  
  - **NP-hard**
    
    - At least as hard has NP, but not necessarily of NP type.

The temporal complexity is an important measurement of the performance of algorithms (e.g. scheduling algorithms)
Scheduling Definition

Task scheduling

- Sequence of task executions in one or more processors
- Application of $R^+$ (time) in $N_0^+$ (task set), assigning to each time instant “t” a task “i” that executes in that time instant.

\[
\sigma: R^+ \rightarrow N_0^+
\]

\[
i = \sigma(t), \ t \in R^+ \quad (i=0 \Rightarrow \text{idle processor})
\]

$\sigma(t)$ is a step function, that has the form of a Gantt graph

$J=\{J_1, J_2, J_3\}$
(task set) →
Scheduling Definition

• A schedule is called **feasible** if it **fulfills** all the **task requirements**
  - **temporal**, non-preemption, shared resources, precedences, ...

• A task set is called **schedulable** if there is **at least one feasible schedule** for that task set
The scheduling problem
(easy to formulate, hard to solve)

• Given:
  - A **task set**
  - **Requirements** of the tasks (or cost function)

• **Find a time attribution of processor(s) to tasks** so that:
  - Tasks are completely executed, and
  - **Meet their requirements** (or minimize the cost function)

\[ J = \{ J_i (C_i=1, a_i=1, D_i=5, i=1..5) \} \]

\[ \sigma(t) \]

\[\begin{array}{c|c}
1 & 2 \\
3 & 4 \\
5 &
\end{array}\]
Scheduling problem

- Build a Gantt diagram of the execution of the following periodic tasks, admitting $D_i = T_i$ and no preemption.
  - $\tau = \{(1,5)(6;10)\}$
- Is the execution order important? Why?
Scheduling algorithms

- A scheduling algorithm is a method for **solving** the scheduling problem.
  - **Note:** don't confuse scheduling algorithm (the process/method) with schedule (the result)

- **Classification of scheduling algorithms:**
  - **Preemptive vs non-preemptive**
  - **Static vs dynamic**
  - **Off-line vs on-line**
  - **Optimal vs sub-optimal**
  - With **strict guarantees vs best effort**
Basic algorithms

**EDD - Earliest Due Date** (Jackson, 1955)

- Single instance tasks fired synchronously:
  \[ J = \{ J_i (C_i, (a_i=0), D_i) \mid i=1..n \} \]

- Executing the tasks by **non-decreasing deadlines minimizes the maximum latency**
  \[ L_{\text{max}} (J) = \max_i (f_i - d_i) \]

- Complexity: \( O(n \cdot \log(n)) \)

\[ \text{e.g. } J = \{ J_1(1,5), J_2(2,4), J_3(1,3), J_4(2,7) \} \]

\[ L_{\text{max, EDD}} (J) = -1 \]
**Basic algorithms**

**EDF - Earliest Deadline First** (Liu and Layland, 1973; Horn, 1974)

- Single instance or periodic, asynchronous arrivals, preemptive: 
  \[ J = \{ J_i (C_i, a_i, D_i) \mid i=1..n \} \]

- Always **executing** the task with **shorter deadline** minimizes the maximum latency 
  \[ L_{\text{max}}(J) = \max_i (f_i - d_i) \]

- Complexity: \( O(n \log(n)) \), **Optimal** among all scheduling algorithms of this class

\[ L_{\text{max},EDF}(J) = -2 \]
**Basic algorithms**

**BB – Branch and Bound** (Bratley, 1971)

- Single instance or periodic tasks, asynchronous arrivals, non-preemptive: 
  \[ J = \{ J_i (C_i, a_i, D_i) \ i=1..n \} \]

- Based on building an exhaustive search in the permutation tree space, finding all possible execution sequences:

- Complexity: \( O(n!) \)

\[ e.g. \ J = \{ J_1(1,0,5), J_2(2,1,3), J_3(1,2,4), J_4(2,1,7) \} \]
Periodic task scheduling

The release/activation instants are known a priori
\[ \Gamma = \{ \tau_i (C_i, \Phi_i, T_i, D_i, i=1..n) \} ; \quad a_{i,k} = \Phi_i + (k-1)T_i, k=1,2,... \]

Thus, in this case the schedule can be built:

- **With the system executing (on-line)**
  Tasks to execute are selected as they are released/finished, during normal system operation

- **Before the system enters in execution (off-line)**
  The task execution order is computed before the system enters in normal operation and stored in a table, which is used at execution time to execute the tasks (**static cyclic scheduling**).
Static cyclic scheduling

- The table is organized in **micro-cycles (μC)** with a fixed duration. This way it is possible to release tasks periodically.
- The micro-cycles are triggered by a **Timer**.
- Scanning the whole table repeatedly generates a periodic pattern, called **macro-cycle (MC)**.

\[ \Gamma = \{ \tau_i (C_i, \Phi_i, T_i, D_i, \ i=1..n) \} \]

\[
\muC = GCD(T_i) \quad (GCD) \\
MC = MCM(T_i) \quad (LCM)
\]

\[ \Phi_i = 0, C_i = 1\text{ms}, \]
\[ T_1 = 5\text{ms}, \]
\[ T_2 = 10\text{ms}, \]
\[ T_3 = 15\text{ms} \]
Static cyclic scheduling

Pros

- Very simple implementation (timer+table)
- *Execution overhead* very low (simple dispatcher)
- Permits complex optimizations
  (e.g. jitter reduction, check precedence constraints)

Cons

- Doesn't scale (changes on the tasks may incur in massive changes on the table. In particular the table size may be prohibitively high)
- Sensitive to overloads, which may cause the “domino effect”, i.e., sequence of consecutive tasks failing its deadlines due to a bad-behaving task.
How to build the table:

- Compute the micro and macro cycles (μC and MC)
- Express the periods and phases of the tasks as an integer number of micro-cycles
- Compute the cycles where tasks are activated
- Using a suitable scheduling algorithm, determine the execution order of the ready tasks
- Check if all tasks scheduled for a given micro-cycle fit inside the cycle. Otherwise, some of them have to be postponed for the following cycle(s)
- It may be necessary to break a task into several parts, so that each one of them fits inside the respective micro-cycle
Summary of Lecture 5

• The concept of **temporal complexity**
• Definition of **schedule** and **scheduling algorithm**
• Some basic **scheduling techniques** (EDD, EDF, BB)
• The **static cyclic scheduling** technique